Inhomogeneous Force-Directed Layout Algorithms in the Visualisation Pipeline: From Layouts to Visualisations

Neville Churcher **Warwick Irwin** **Carl Cook**

Software Visualistion Group, Department of Computer Science,
University of Canterbury, Private Bag 4800, Christchurch, New Zealand
E-mail: {neville,wal,c.cook}@cosc.canterbury.ac.nz

Abstract

The visualisation pipeline approach is a flexible and extensible technique for generating visualisations. The basic pipeline functions involve the capture and representation of data, the computation of geometry and the presentation of visual output. Data is represented in XML at each stage and is successively transformed, typically by XSLT transformations, as it moves through the pipeline. A force-directed layout engine, ANGLE is one of the major components in the computation of 2D and 3D geometry. In this paper, we describe inhomogeneous force models and their implementation in ANGLE. These allow a richer variety of properties and relationships of the underlying graph and application domain to be included in the visualisation. We present examples from our software visualisation research.

1 Introduction

Force-directed layout techniques, also known as spring embedders, (Eades 1983, Eades 1984, Di Battista, Eades, Tamassia & Tollis 1999) are a reliable general-purpose tool for graph layout applications. Although other techniques may be capable of delivering superior results for particular classes of graphs, force-directed methods have found application in a wide range of areas.

In our work, we require layout algorithms which can reliably provide a satisfactory layout when presented with an unknown graph. It is more important that our layouts be "good enough all the time" than "great some of the time". Force-directed methods generally exhibit smooth convergence to the final layout configuration, making them particularly suitable for layouts which are presented dynamically to users via a graphical user interface.

The remainder of the paper is structured as follows. In the next section, we outline our visualisation pipeline approach and describe the rôle of ANGLE (Churcher & Creek 2001), our layout engine. We introduce inhomogeneous force models in Section 3 and discuss our approach and its implementation in Section 4. We illustrate our approach by presenting some examples of its application in Section 5. Finally, our conclusions and discussion of work in progress are presented in Section 6.

2 Layout in the visualisation pipeline

The conventional visualisation pipeline (Schroeder, Martin & Lorensen 1998, for example) consists of three basic phases: data capture/generation & processing; computation of geometry and rendering of the resulting visualisation.

Our recent work in information and software visualisation (Churcher, Keown & Irwin 1999, Irwin & Churcher 2002, Churcher, Irwin & Kriz 2003, Irwin & Churcher 2003) involves a flexible extensible pipeline approach to information and software visualisation.

The pipeline, shown schematically in Figure 1, consists of three main sections, each of which will be discussed further below.

XML (Birbeck, Diamond, Duckett, Gudmundsson, Kobak, Lenz, Livingstone, Marcus, Mohr, Pinnock, Visco, Watt, Williams, Zaev & Ozu 2001, Sall 2002, W3C n.d.) is used to represent the data at each stage. As data passes through the pipeline is it transformed in various ways. In most cases, XSLT transformations are sufficient. When significant processing is required, as in a semantic analyzer or a graph layout tool, filters written in any language (usually Java in our case) may be readily inserted into the pipeline.

In Figure 1, the data capture section is indicated by a light grey background. In a typical software visualisation context, this section involves parsing source code files to obtain individual syntax trees. These are then combined using a semantic analyzer in order to resolve larger scale issues such as method overriding and invocation. The resulting model may then be filtered to remove elements which will not participate further in the visualisation process. The parsers are produced with the aid of yakyacc (yet another kind of yacc) which uses XML pipeline-based technology to generate parsers from standard programming language grammars. The semantic analysis tools make explicit the semantic structures defined by the corresponding language standards. Both yakyacc and the semantic analysis tools are described in more detail elsewhere (Irwin & Churcher 2002, Irwin & Churcher 2003).

In simpler application contexts, some components may be omitted from this pipeline section (an example is given in §5.1. For example, a semantic analyser was not required in our single class cohesion visualisations (Churcher et al. 2003). In the simplest cases, such as graph layout, this section reduced to a simple file which may be created with the aid of an appropriate application or simply edited by hand.

At the end of this pipeline section, the model consists of an XML representation of the original data in the vocabulary of the original domain.

The geometry computation section is indicated by a mid-grey background.

In our case, geometry computation typically involves ANGLE, our force-directed layout

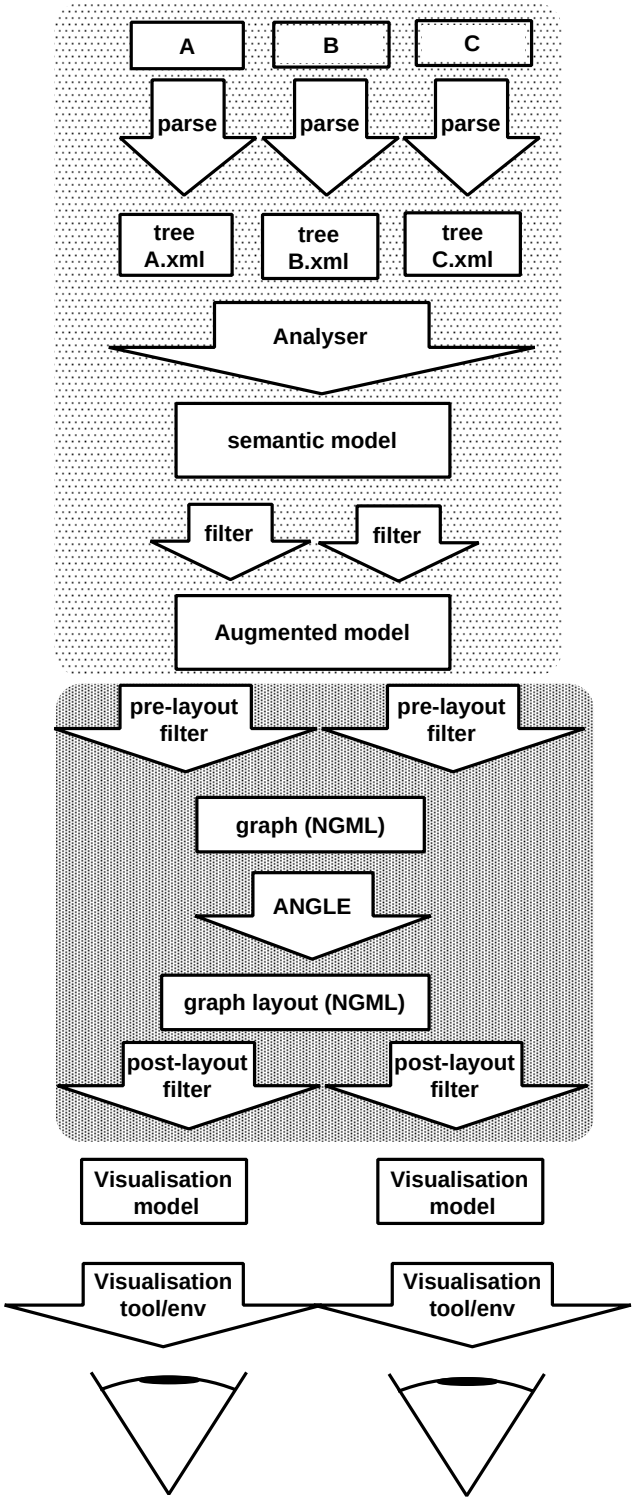


Figure 1: Visualisation pipeline

tool (Churcher & Creek 2001). Given an XML representation of a graph, in a geometrical vocabulary involving terms such as nodes and edges, ANGLE can produce 2D or 3D layouts—effectively adding spatial data to the XML format. ANGLE's architecture allows new force models and terminators to be defined as required.

Designing a visualisation involves establishing mappings from the elements of the application domain represented in the data model to those of the geometric model (e.g. classes will be represented by nodes in a graph and edges will represent inheritance) and from the geometric model to the presentation model (e.g. class nodes will be represented by unlabelled red spheres whose radius is proportional to the number of methods). These mappings are achieved via the use of pre-layout and post-layout filters (Churcher et al. 2003).

Pre-layout filters may be inserted in this pipeline section to perform transformations which select from the data model those elements which will participate in the layout process and convert them to appropriate structure and naming conventions. Domain specific data, which is required for subsequent mappings but might not play any rôle in layout computation, is transformed into (XML) attributes of the corresponding geometric elements by pre-layout filters. For example, the name of each class and its number of methods might become attributes of the corresponding graph node.

After the layout has been performed (i.e. node positions determined) post-layout filter pipeline components may be used to utilize the domain data which has been added by pre-layout filters. This information is then available for use in constructing the presentation mappings. For example, the class name might appear in a label, and the number of methods might be used to determine the radius of the corresponding symbol.

Examples of this process appear in Section 5.

3 Inhomogeneous spring models

Force-directed layout algorithms(Eades 1983, Eades 1984, Di Battista et al. 1999) are widely used to obtain 2D and 3D layouts for undirected graphs. The approach involves modelling pairwise repulsive forces $\mathbf{F}^r$ between nodes and attractive forces $\mathbf{F}^a$ between nodes connected by edges. The functional forms of $\mathbf{F}^a$ and $\mathbf{F}^r$ are arbitrary and the layout represents an equilibrium solution obtained by iterative solution of the corresponding equations of motion.

A homogeneous model is commonly assumed. In such a model, the strengths of attractive and repulsive forces depend on the distances between nodes. They do not depend on properties of individual nodes or weights of individual edges other than through global constants. We will first describe the homogeneous model and then consider the inhomogeneous model resulting from relaxation of this constraint.

A common choice of functional form for $\mathbf{F}^a$ is based on Hooke's law (equation 1) which gives the attractive restoring force for elastic springs of natural length l_0 when stretched by displacement x.

$$F^a = -k^a(x - l_0) \tag{1}$$

Real springs are elastic only for small displacements but this requirement can safely be relaxed in our case. Where $l_0 > 0$ a negative value of x will result in a force acting to prevent the further compression; the spring is trying to return to its natural length irrespective of the sign of the extension. The constant k^a represents the stiffness of the spring: the

stiffer the spring, the greater the force required to produce a given extension.

It is common to assume homogeneous values for k^a and l_0 so that each spring has the same stiffness and each has the same natural length.

The functional form for the repulsive force between nodes is commonly based on an inverse square law behaviour (equation 2) where the strength of the repulsive force between nodes i and j falls off in proportion to the square of the distance between them (equation 2). In practice, some slight modification is needed in order to prevent singularities where the distance, r between nodes becomes very small.

It is common to assume homogeneous values for k^r, the constant which determines the strength of the repulsion. Thus, each node exerts a repulsive force on each other node which depends only on their separation.

$$F^r_{ij} = \frac{k^r}{r^2_{ij}} \tag{2}$$

Combining the contributions from all forces in a homogeneous model leads to equation 3 for the total force on node i, obtained by combining the individual contributions from individual pairs of nodes as described by equations 1 and 2. The force and distance quantities are now represented by vectors and $\hat{\mathbf{r}}_{ij}$ denotes the unit (direction) vector between nodes i and j.

$$\mathbf{F}_i = \sum_{i,j \in V, i \neq j} \frac{k^r \hat{\mathbf{r}}_{ij}}{|\mathbf{r}_j - \mathbf{r}_i|^2} + \sum_{i,j \in E, i \neq j} -k^a((\mathbf{r}_j - \mathbf{r}_i) - \frac{l_0(\mathbf{r}_j - \mathbf{r}_i)}{|(\mathbf{r}_j - \mathbf{r}_i)|}) \tag{3}$$

Our mental picture of the homogeneous force directed layout model described thus far involves identically charged particles repelling each other while some pairs of particles are joined by springs of identical natural length and stiffness.

In the inhomogeneous counterpart, which we will now introduce, only slight changes to this picture are required. The particles may carry different amounts of charge: the repulsive force between a pair of nodes will depend on the product of their charges. Similarly, the network of springs connecting nodes may contain some very stiff springs as well as weaker ones and they may have a variety of natural lengths.

In the physical world from which the inverse square behaviour has been borrowed, the repulsive force would correspond to that between charged particles which might carry different charges as described by equation 4.

$$F^r = \frac{1}{4\pi\epsilon_0} \frac{q_i q_j}{r^2_{ij}} \tag{4}$$

In equation 4, q_i and q_j represent the (possibly different) charges on particles (nodes) i and j respectively, while $\frac{1}{4\pi\epsilon_0}$ is constant over the entire system. Similar functional forms apply to other natural inverse square forces such as Newton gravitation, where the masses of particles take the place of charges.

Thus, we can relax the homogeneity aspect of equation 2 by representing the repulsive force between nodes as shown in equation 5.

$$F^r = k^r \frac{q_i q_j}{r^2_{ij}} \tag{5}$$

The inhomogeneous model counterpart of equation 5 is equation 6.

$$\mathbf{F}_i = \sum_{i,j \in V, i \neq j} \frac{k^r q_i q_j \hat{\mathbf{r}}_{ij}}{|\mathbf{r}_j - \mathbf{r}_i|^2} + \sum_{i,j \in E, i \neq j} -k^a_{ij}((\mathbf{r}_j - \mathbf{r}_i) - \frac{l_{0_{ij}}(\mathbf{r}_j - \mathbf{r}_i)}{|(\mathbf{r}_j - \mathbf{r}_i)|}) \tag{6}$$

In the case where all "charges" are equal and all springs are identical, equation 6 reduces to equation 3 as expected.

In our visualisation work, we do not normally encounter "real" charges or stiffnesses. In general, we deal with graphs $G = (N, E)$ whose nodes n_i and edges e_ij each have properties or attributes of their own. For example, each node may represent a product which has properties such as cost, price and weight; similarly edges might represent pipeline segments with attributes such as length, cost and capacity.

The corresponding charges, stiffnesses and natural lengths in the spring model will be determined by mappings based on functions of the attributes of the underlying data. For example, "charge" might be based on $value = \frac{cost}{price}$. The mapping selection used in specific instances will be chosen to reflect the specific visualisation desired.

4 Representing inhomogeneous models

We often need to visualise systems whose component nodes are of different types and which have several kinds of connections. Representing these with a homogeneous model is often problematic.

Consider the familiar example of the entity-relationship model (Chen 1976) used in database schema design.

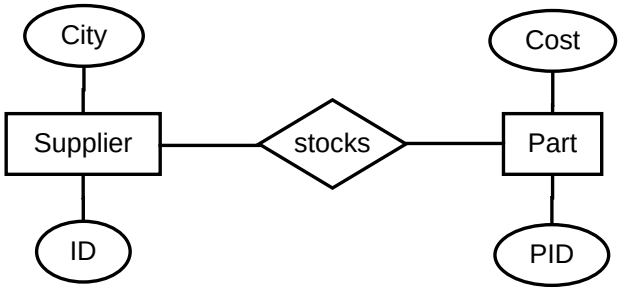


Figure 2: Entity-relationship diagram

Nodes representing entities are "more important" than those representing attributes. Edges connect attributes to their entities and related entities to each other via relationship nodes. In a "nice" layout, each entity's attributes are clustered tightly around it, the entity nodes are spaced further apart and nodes representing relationships are symmetrically placed between the entities they connect.

In the inhomogeneous model, entity and relationship nodes could be encouraged to stay apart by giving them a higher charge, Q_{hi}, than that of the other nodes, Q_{lo}, while keeping all springs identical. There would then be three repulsive interaction strengths, each proportional to the inverse square of the node separation.

The repulsive forces between pairs of entity nodes would then be proportional to ${Q^2}_{hi}$, as would the forces between entity nodes and relationship nodes;

the force between an attribute node and an entity node would be proportional to $Q_{hi} \times Q_{lo}$, as would the force between an attribute node and a relationship node; the forces between pairs of attribute nodes would be proportional to ${Q^2}_{lo}$.

Alternatively, we could keep all charges identical and use shorter, stiffer springs for entity-attribute edges than for entity-relationship edges.

In general, there are many combinations of charge, natural length and stiffness which would encourage the same behaviour. The interactions between various combinations are not always straightforward to anticipate.

We have developed an inhomogeneous force model to add to those supplied with ANGLE. This model is an inhomogeneous version of the "Big Bang" model and will be referred to as the BBC (Big Bang + Coulomb) model hereafter. Figure 4 shows the controls for the BBC algorithm. These include sliders for setting the global values of stiffness (ka), natural length (l0) and charge (Q). Default values for all algorithm parameters are provided.

The default values may be overridden for individual nodes and edges in the the NGML data files as shown in Figure 3. We have chosen to represent the various parameters as multipliers of a global default rather than absolute values. This allows the interactive controls to be used more conveniently. The node elements have an optional `charge` attribute which represents a value in terms of how many units of the current global default charge apply. Similarly, the edge elements have optional `stiffness` and `l0` attributes which also indicate multipliers for the current global settings of the corresponding parameters.

```
...
<node charge="2.0">
  <name>nA</name>
</node>
...
<edge stiffness="0.1" l0="2">
  <from>nA</from>
  <to>nB</to>
</edge>
<edge stiffness="3">
  <from>nA</from>
  <to>nC</to>
</edge>
<edge l0="5">
  <from>nA</from>
  <to>nD</to>
</edge>
...
```

Figure 3: Representing inhomogeneous model data in NGML

5 Applications

In this section, we illustrate the application of inhomogenoeus force models with some specific examples in order to demonstrate the rôle of inhomogeneous force models in producing visualisations. The presentation format is VRML (Carey & Bell 1997) delivered to users via browser plug-ins such as cortona (`http://www.parallelgraphics.com`. Printed greyscale images do not convey the full impression of these worlds—please visit our web page (svg 2000) to experience the interactive colour versions for yourself.

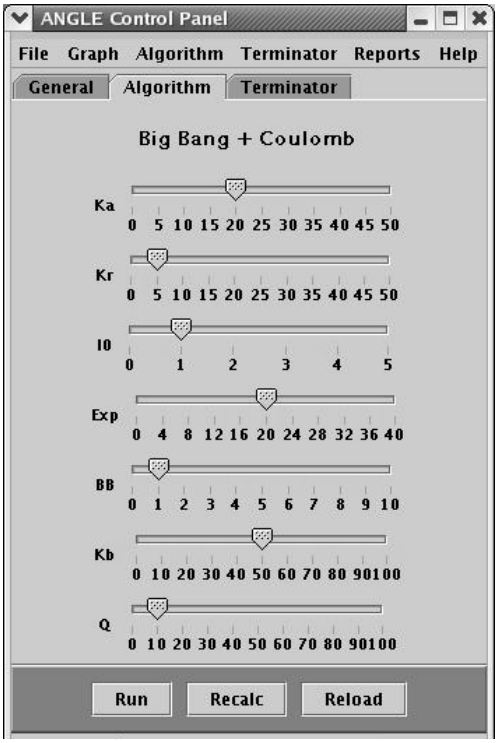


Figure 4: ANGLE inhomogeneous model algorithm controls

5.1 Plagiarism detection

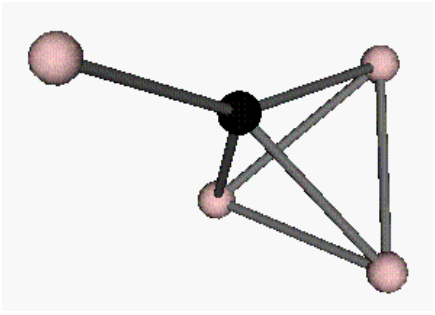
Our student assignments are routinely compared by plagiarism detection software. The plagiarism detection software compares each student assignment with that of every other student and expresses the result as a percentage. False positives (e.g. those arising where students simply submit unmodified versions of the template code skeleton provided) are then removed and the remainder are sorted in decreasing order of similarity. Typically, the degree of similarity falls off rapidly and then stabilises at a level of "normal" similarity representing the expected degree of similarity between any two independently-developed solutions to the same problem.

Effective visualisations of the results of such comparisons are a key factor in identifying suspicious similarities for further investigation. One natural metaphor is to represent the similarity network by a (3D) graph. While statistical clustering isn't (we hope!) applicable, the idea is that connected "clusters" of nodes will correspond to groups of collaborating students. Unfortunately, simply adding edges in decreasing order of similarity (i.e. including only edges corresponding to similarities greater than some threshold value) leads to graphs with many disconnected components. Not only does this cause distress to force-directed layout algorithms—it also obscures some of the important relationships *between* clusters. Cases of plagiarism often involve only two students but sometimes there is a link between clusters. This could involve a common acquaintance but in some cases arises from a hard-copy listing being stolen from a printer.

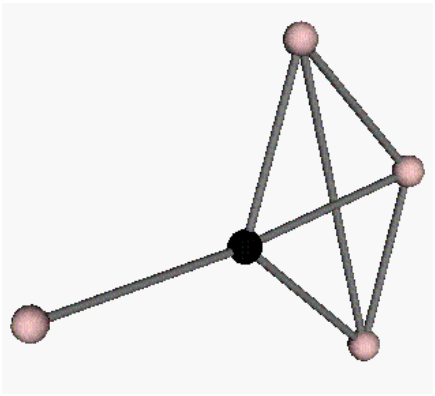
Figure 5 shows two screen snapshots of a common scenario possibly involving a group of four collaborators and a separate collaboration with one of these (the darkest node). Only edges where similarity is greater than a threshold value are included.

In both cases, the edge stiffness is related to the observed similarity. Figure 5(a) shows a linear relationship where the stiffness is proportional to the

percentage similarity. Our experience suggests that the final layout is more sensitive to non-linear mappings such as those used to generate Figure 5(b). The "weakest link" has stretched, allowing the tight tetrahedral arrangement to flatten. The interpretation, supported by inspection, is that the darkest node represents the master-criminal, who has collaborated independently with each of the others.



(a) Stiff



(b) Relaxed

Figure 5: Small candidate plagiarism cluster

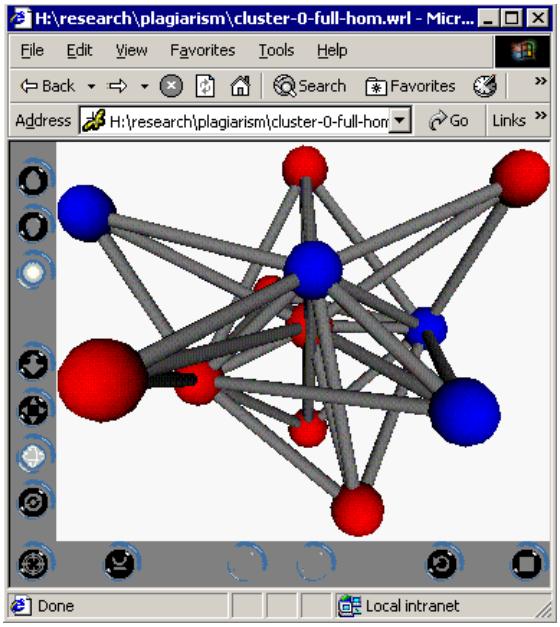
As the threshold is lowered, more edges are added and the graph becomes more highly connected. However, the likelihood of missing relevant connections because of the arbitrarily chosen threshold decreases.

Figure 6 shows another example cluster from the same data set. Figure 6(a) shows a layout resulting from a homogeneous algorithm and the browser controls used for navigating the resulting world. Post-layout filters have been applied to dress the bare graph with node labels and node and edge colours as they appear in Figures 6(b) and 6(c).

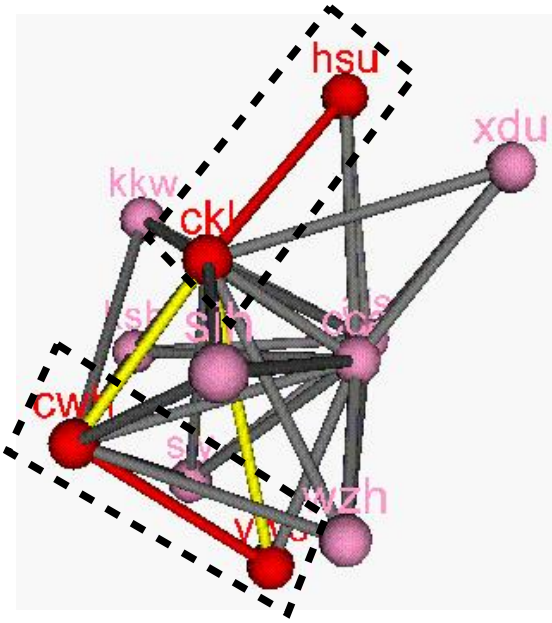
Figure 6(b) results from a linear mapping of spring stiffness to edge similarity. The raw similarity data suggests two likely collaborations each involving a pair of students: these are indicated by the dashed rectangles in Figure 6(b). Post-layout filters can draw attention to edges corresponding to high similarity even though they are not immediately obvious in the layout.

While this is an improvement over the homogeneous layout, the range of stiffnesses resulting from the simple linear mapping alone is not sufficient to reveal all the desired information. Figure 6(c) results from a mapping using "binned" similarity values to derive stiffnesses plus allowing the l_0 value for individual edges to reflect the similarity (i.e. edges representing strong similarity are initially shorter and stiffer than those representing more tenuous similarity values.

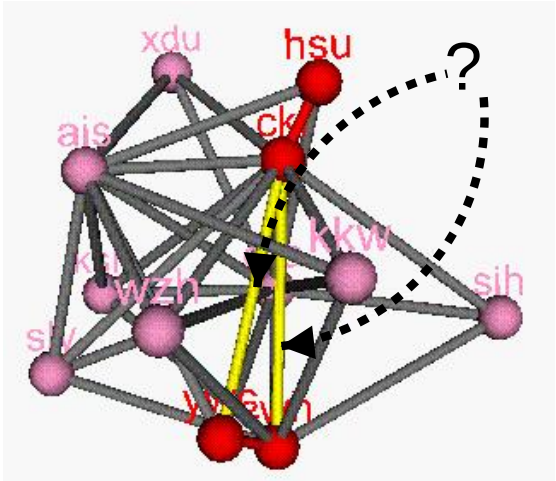
In Figure 6(c) the suspicious pairs have moved closer together and some "interesting" edges (indicated by dotted arrows) between them are now more



(a) Homogeneous



(b) Inhomogeneous



(c) Inhomogeneous + l_0

Figure 6: Plagiarism examples

readily apparent. Edge colour indicates similarity value and these edges correspond to a connection between the two pairs which would have been hard to detect without the visualisation.

The node labelled 'xdu' has a mild similarity to only one of the suspicious nodes at this threshold level and is thus likely to be "innocent". However, node 'ajs' has many similarities and warrants a closer inspection.

5.2 Pre-requisite structure

To illustrate the use of filters in combination with the BBC inhomogeneous layout model we will use the example of the pre-requisite relationships between courses in our department. Courses have a credit point value (3 or 6) and may be full-year or single semester (first or second) in duration. There are two kinds of relationships between courses: a course may be a required pre-requisite for other courses, or may be recommended preparation for other courses.

```
<node points="6" semester="f">
  <name>121</name>
</node>
<node points="3" semester="s">
  <name>226</name>
</node>
...
<edge kind="p">
  <from>121</from><to>226</to>
</edge>
<edge kind="r" stiffness="0.01">
  <from>225</from><to>314</to>
</edge>
...
```

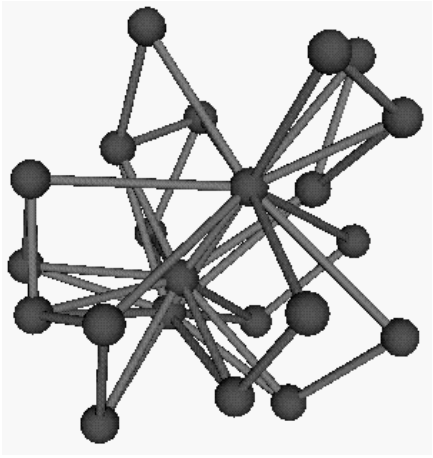
Figure 7: Representing course structure

Figure 7 shows how the domain attributes `points` and `semester` have been included in the node element while edges carry an attribute indicating the `kind` of constraint between courses they represent. This is an example of the situation described, in section 2, where the data capture section of the pipeline reduces to data file creation.

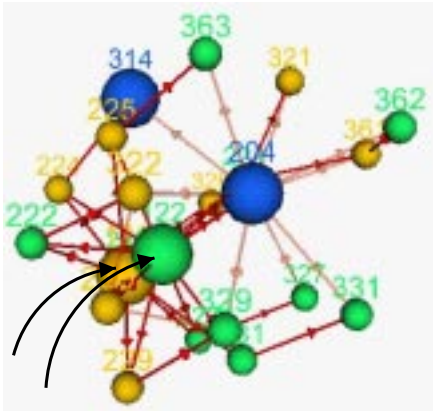
Figure 8 shows screen snaps of four layouts based on the data described in Figure 7. Figure 8(a) shows the "bare" graph layed out using the homogeneous model with default values for all parameters. Figure 8(b) shows the same layout after a post-layout filter has been applied. The filter, implemented as an XSLT transformation, has used the domain attribute values, which took no part in the layout, to decorate the bare graph.

Node radius is proportional to credit point value, node colour indicates semester and course codes appear as labels. Edge colour and transparency indicate the kind of constraint they represent and arrow heads have been added to indicate the direction of the constraint. Force-based layout algorithms generally assume undirected graphs.

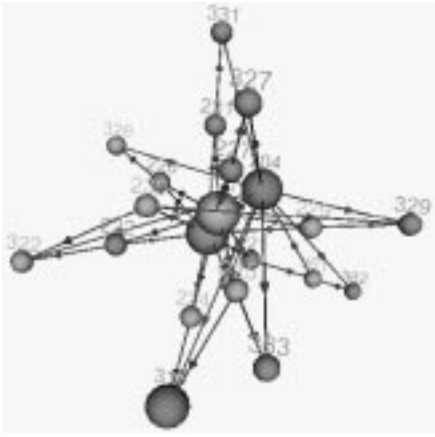
We contend that Figure 8(b) is clearly a superior visualisation, though the underlying graph is identical to that of Figure 8(a). A number of features are evident. The two 100-level (first year) courses, 121 & 122 (identified by the arrows in Figure 8(b)), are both pre-requisites for each 200-level course and hence are located close to each other. The 204 course is recommended preparation for every 300-level course but is a pre-requisite for none. In most cases a 2xx course is a pre-requisite for a corresponding 3xx course in the same field. Figure 8(b) contains several 204, 2xx, 3xx



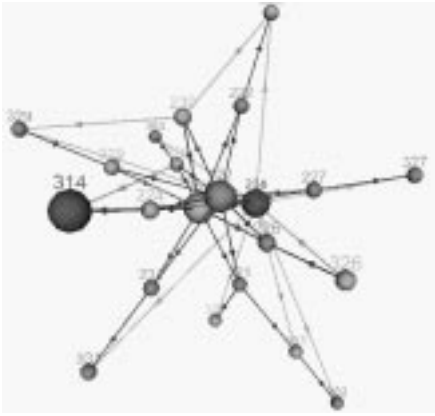
(a) Bare graph (homogeneous)



(b) 8(a) styled with filters



(c) Inhomogeneous model $l_0 = 1$



(d) Inhomogeneous model $l_0 = 2$

Figure 8: Visualising course structure

triangles which somewhat obscure this fundamental relationship.

Using an inhomogeneous model overcomes this problem. Figure 8(c) shows the layout resulting from the lowering of the stiffness of edges corresponding to the weaker recommended preparation constraint to a tenth of the strength of the others (as indicated in Figure 7). This allows the stronger pre-requisite constraints to dominate and the overall effect becomes one with radial (121&122) $\rightarrow$ $2xx$ $\rightarrow$ $3xx$ strands corresponding to various specialisations such as databases.

Finally, Figure 8(d) shows the effect of doubling the global l_0 value. The resulting graph is larger, but the essential features are similar.

5.3 Class clusters

A common problem in our software visualisation research is the representation of the relationships between components of object-oriented software.

The application domain involves components of several kinds (applications, libraries, packages, classes, interfaces, methods, data items, control constructs, ...) and a number of different kinds of connections between them (inheritance, membership, composition, invocation, overloading, ...). Both components and connections have further attributes: classes have names and methods have return types; membership has an access mode (public, protected, private) and invocation involves caller and callee methods.

The number of possible visualisations based on such a model is huge and it is not realistic to expect to show all relationships simultaneously. Our pipeline-based approach allows users to design and generate a custom visualisation with a modest amount of effort.

Some components and relationships are intrinsically more "important" than others. This is reminiscent of the concept of *degree of interest* (DOI) used in distortion-oriented (fisheye) visualisation (Furnas 1986, Furnas 1997, Sarkar & Brown 1994). The DOI includes an *a priori interest* (API) contribution, reflecting the fact that some visualisation elements are intrinsically more interesting. For example, classes may be deemed to be more important than methods in some visualisations. The DOI also includes a *distance* contribution reflecting how far away—in conceptual or Euclidean space—components are.

Combining these contributions leads to a situation where both context and detail can be represented. In our application, we may wish to focus on an individual class (i.e. move close to it in our virtual world) yet still be aware of the most "relevant" classes in the neighbourhood. Such a "gestalt" view of OO software is important to software engineers.

Class clusters highlight aspects of the "neighbourhood" of individual classes (those related by inheritance or method invocation and hence likely to be considered together during development) while also showing large-scale structure of a system (a single package in our example). The idea is that strongly-related classes should be physically close to each other in a manner reminiscent of statistical clustering (Everitt 1993).

Figure 9 shows a class cluster corresponding to a Java package containing 23 classes having 153 methods. The edges of the graph represent relationships between the elements mapped to nodes: 19 inheritance connections, 153 class-has-method occurrences and 81 method invocations.

The pre-layout filter stage has selected a subset of the data available from the semantic analyzer for inclusion as domain attributes. Domain attributes not selected in this example include those representing aggregation relationships and visibility and hence these factors do not contribute to the layout process. However, no visualisation can effectively convey all relevant variables simultaneously. Our approach enables the user to select and modify filters at will and hence greatly enhances its usefulness in exploratory analysis.

The layout results from use of the BBC algorithm as described below, and an XSLT post-layout filter has assigned properties such as node colours and edge thicknesses.

- Each class is represented by a (red) sphere whose radius is proportional to the number of methods (public, locally-defined, excluding constructors) in the class.

- The sphere representing each class node is studded with smaller (blue) spheres representing its methods. At the core of the class node is a node to which each method is connected by a *class-has-method* edge (not visible in Figure 9 because they lie entirely within class spheres which the post-layout filter mappings made opaque).

  The nodes corresponding to methods are subject to repulsive forces from all other nodes and attractive forces via invocation edges in addition to those anchoring them to their own class. Thus they are constrained to move only on the surface of the corresponding class sphere in order to align as closely as possible to methods which they invoke.

  In order to ensure that the spheres representing methods remain on the surface of the corresponding class sphere the *class-has-method* edges are made stiff (stiffness="2") and their l_0 values are assigned to be the number of methods in the corresponding class. Thus, these edges are very rigid.

- Inheritance edges are denoted by (red) edges with arrows pointing to the superclasses. In Figure 9, the thickness of inheritance edges indicates the *number of leaf classes supported* (NLCS) metric.

  Inheritance edges are represented by much slacker, longer springs (stiffness="0.2", l0="50") since we wish the separation of classes to be greater than typical class size.

- Invocation of methods is indicated by the thinner (yellow) lines connecting methods. The edge direction (caller to callee) is not shown but this information is available in the model.

  Invocation edges are the slackest of all (stiffness=".005"). This is because we want the edge structure to be determined primarily by the inheritance structures. In the absence of any invocation edges, the root class would be at the centre of a spherically symmetric cluster.

Figure 10 illustrates the effect of customising the filters used in the pipeline. It shows two ways of adding the method level metrics cyclomatic complexity (ν) and number of statements (STMT) to the class cluster visualisation. Figure 10(a) results from mappings which represent methods by cones whose base radii represents ν and whose heights represent STMT values. The user can identify outliers by their aspect ratios. In Figure 10(b), the mappings used include more specific criteria: cubes denote methods where $\frac{STMT}{\nu} \geq 2$ and darker colour denotes lower ν values.

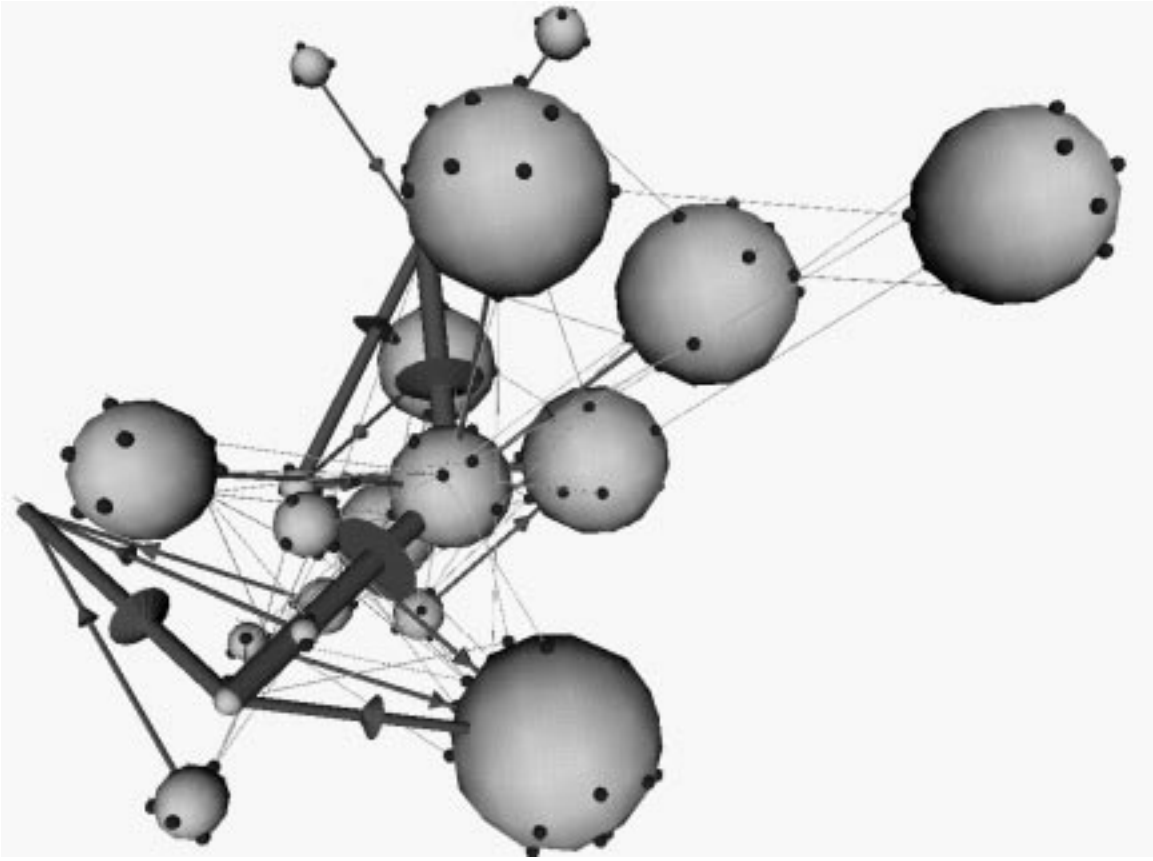
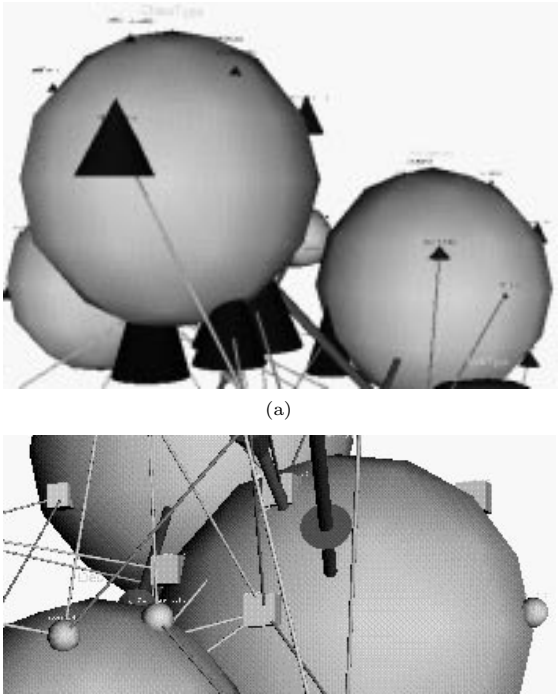


Figure 9: Class cluster



(a)

(b)

Figure 10: Inclusion of method metrics

6 Conclusions & future work

In this paper we have reported the extension of our Angle layout tool to include inhomogeneous force models. Our XML-based pipeline approach to visualisation allows layout tools such as Angle to be inserted at the stages where geometry is computed. The pipeline approach includes pre-layout filters, which control the information available to the layout process, and post-layout filters, which dress the bare computed geometry in order to realise the desired mappings of domain properties to characteristics of the rendered visualisation. Domain information, together with graph structure information, is available in the pipeline and appears as node values or edge weights.

When combined with pre-layout and post-layout filters, inhomogeneous force models provide a powerful tool for producing geometry which includes domain data in a natural way. This approach is particularly valuable in situations where nodes and edges of more than one kind are present.

We have presented some applications to illustrate how the use of inhomogeneous force models allows richer and arguably more effective visualisations to be constructed with a minimum of user intervention.

Our work in progress is following a number of threads. We are continuing to apply inhomogeneous models in software visualisation. We are attempting to develop new metaphors, such as the class clusters described here, and to extend existing metaphors. The aim is to help software engineers to comprehend, develop and maintain large software systems.

Another ongoing aspect of our work is concerned with improving the authoring tools available to users to allow exploration of the effects of different force model choices. For example, one may need to decide whether to represent a domain interaction via node

charges or spring stiffnesses.

Evaluating the computational efficiency effects of parameter choices is also important. In general, for a given graph, a homogeneous force model will terminate after fewer iterations than an inhomogeneous force model. This is understandable because of effects of inhomogeneous model parameter settings, such as increasing charges on nodes in a given configuration, acting to increase the total energy of a given configuration. Our experience suggests that the increase in required number of iterations are acceptable. However, we would like to be able to quantify the effects more precisely.

User feedback and anecdotal evidence and concerning the value of visualisations including inhomogeneous models is encouraging. However, as our approach matures, it will ultimately be necessary to conduct more formal user trials in order to help determine the relative merits of particular visualisations in specific contexts.

References

Birbeck, M., Diamond, J., Duckett, J., Gudmundsson, O., Kobak, P., Lenz, E., Livingstone, S., Marcus, D., Mohr, S., Pinnock, J., Visco, K., Watt, A., Williams, K., Zaev, Z. & Ozu, N. (2001), *Professional XML*, 2nd edn, Wrox Press.

Carey, R. & Bell, G. (1997), *The Annotated VRML 2.0 Reference manual*, Addison-Wesley.

Chen, P. P.-S. (1976), 'The entity relationship model—towards a unified view of data', *ACM Transactions on Database Systems* **1**(1), 9–36.

Churcher, N. & Creek, A. (2001), Building virtual worlds with the big-bang model, *in* P. Eades & T. Pattison, eds, 'Information Visualisation 2001', Vol. 9 of *Conferences in Research and Practice in Information Technology*, ACS, Sydney, Australia, pp. 87–94.

Churcher, N., Irwin, W. & Kriz, R. (2003), Visualising class cohesion with virtual worlds, *in* T. Pattison & B. Thomas, eds, 'Australasian Symposium on Information Visualisation, (invis.au'03)', Vol. 24 of *Conferences in Research and Practice in Information Technology*, ACS, Adelaide, Australia, pp. 89–97.

Churcher, N., Keown, L. & Irwin, W. (1999), Virtual worlds for software visualisation, *in* A. Quigley, ed., 'SoftVis99 Software Visualisation Workshop', University of Technology, Sydney, Australia, pp. 9–16.

Di Battista, G., Eades, P., Tamassia, R. & Tollis, I. G. (1999), *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall.

Eades, P. (1983), A heuristic for graph drawing, *in* D. S. Meek & G. H. J. v. Rees, eds, 'Proc. 13th Manitoba Conf. Numerical Mathematics and Computing', Utilitas Mathematica Publishing, Winnipeg, Canada.

Eades, P. (1984), 'A heuristic for graph drawing', *Congressus Numerantium* **42**, 149–160.

Everitt, B. (1993), *Cluster Analysis*, 3rd edn, Edward Arnold.

Furnas, G. (1986), Generalised fisheye views, *in* 'Proc ACM SIGCHI '86 Conference on Human Factors in Computing Systems', pp. 16–23.

Furnas, G. (1997), Effective view navigation, *in* S. Pemberton, ed., 'CHI 97', Atlanta, GA, pp. 367–374.

Irwin, W. & Churcher, N. (2002), XML in the visualisation pipeline, *in* D. D. Feng, J. Jin, P. Eades & H. Yan, eds, 'Visualisation 2001', Vol. 11 of *Conferences in Research and Practice in Information Technology*, ACS, Sydney, Australia, pp. 59–68. Selected papers from 2001 Pan-Sydney Workshop on Visual Information Processing.

Irwin, W. & Churcher, N. (2003), Object oriented metrics: Precision tools and configurable visualisations, *in* 'METRICS2003: 9th IEEE Symposium on Software Metrics', IEEE Press, Sydney, Australia, pp. 112–123.

Sall, K. B. (2002), *XML Family of Specifications: A Practical Guide*, Addison-Wesley.

Sarkar, M. & Brown, M. (1994), 'Graphical fisheye views', *Communications of the ACM* **37**(12), 73–84.

Schroeder, W., Martin, K. & Lorensen, B. (1998), *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, 2nd edn, Prentice Hall.

svg (2000), 'Software Visualisation Group, Department of Computer Science, University of Canterbury', `http://www.cosc.canterbury.ac.nz/research/RG/svg`.

W3C (n.d.), 'World wide web consortium', `http://www.w3c.org`. Cited 2003.